

Sergei Ossif

Applications of FPGAs in high-performance adaptive channel equalization

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Degree Programme in Electronics

Bachelor's Thesis

02.02.2016

Author(s)	Sergei Ossif
Title	Applications of FPGAs in high-performance adaptive channel equalization
Number of Pages	24 pages + 1 appendices
Date	Tuesday 2 nd February, 2016
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Electronics
Specialisation option	
Instructor(s)	Thierry Baills, Senior Lecturer
<p>Wireless communications play an important role in today's society. To achieve higher speeds and bandwidth efficiency modulation techniques moved into the digital domain. The implementation relies on the use of digital signal processing to reduce intersymbol interference.</p> <p>The idea of this project was to investigate the feasibility of building such system on FPGA platform. An implementation of Fractionally Spaced Equalizer using Least Mean Squares filters to configure 8-tap linear filters was built and evaluated. To test the system, the equalizer was connected to a microprocessor. The processor then fed the test sequence and calculated the time it takes to process the data. The results were then compared to a similar system built on DSP multicore processor system.</p> <p>This implementation showed promising results. The system showed itself to be about 40 times faster than the DSP multicore processor system with an average time required to process 64 samples to be 13.7 μs.</p>	
Keywords	Digital Signal Processing, DSP, Least Mean Squares, LMS, Field-Programmable Gate Array, FPGA, Adaptive Filtering, Adaptive Equalizer, Quadrature Amplitude Modulation, QAM

Contents

List of Figures	4
Abbreviation	
1 Introduction	1
2 Theoretical Background	2
2.1 Transmission	2
2.1.1 Representation of digitally modulated signals	3
2.1.2 Quadrature amplitude modulation	4
2.2 Channel	5
2.2.1 Rayleigh Fading Channel	7
2.3 Reception	10
2.3.1 Demodulator	10
2.3.2 Equalizer	10
2.3.3 Least Mean Squares filter	12
3 Implementation	15
3.1 Workflow	15
3.2 Hardware	15
3.3 Software	16
3.4 Implementation	18
4 Results	21
Bibliography	23
Appendices	
Appendix 1 Benchmark Code	

List of Figures

1	A simple model of a communications system (reproduced from [1]).	2
2	Several signal space diagrams(reproduced from [2]).	4
3	Relationships between the channel frequency-transfer function and a signal with bandwidth W (reproduced from [3]).	6
4	Multipath propagation (copied from [4]).	7
5	Small-scale fading: mechanisms, degradation categories, and effects (reproduced from [3]).	8
6	Fading channel manifestations (reproduced from Proakis [3]).	9
7	Linear and decision feedback equalizers (reproduced from [1]).	11
8	Response of a multipath channel to a narrow pulse vs. delay, as a function of antenna position (reproduced from [3]).	12
9	Adaptive LMS filter (copied from [5])	13
10	Basic DSP48E1 Slice Functionality (copied from [6]).	16
11	Complete system	17
12	Top view of System Generator	18
13	Fractionally spaced equalizer	19
14	Adaptive Filter top view	20
15	LMS Filter	20
16	Simplified diagram of the test environment	21
17	Equalizer error at $\mu=0.05$	21

Abbreviations

AWGN	Additive White Gaussian Noise
DFE	Decision Feedback Equalizer
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field-Programmable Gate Array
FSE	Fractionally Space Equalizer
ISI	Intersymbol Interference
LMS	Least Mean Squares
QAM	Quadrature Amplitude Modulation
RISC	Reduced Instruction Set Computer

1 Introduction

In today's world wireless communications take larger and larger role. It is possible now to carry a device capable of transferring information at speeds of up to 1 Gbit/s. With the always increasing requirements to bandwidth speed engineers have to squeeze every single bit of performance out of the hardware to stay competitive. This means innovation in more bandwidth efficient modulation methods, increasing the base frequency, more complex modulators, and demodulators.

With technological limits pushed up so high, higher sensitivity is required. As the sensitivity goes higher, so does the degenerative effects of the environment where the communication takes place. All kinds of objects and weather effects in between of the transmitter and the receiver could inject the noise into the received signal. As the result, more thought is being put into the design of the demodulator.

An equalizer is the part of the demodulator designed to battle these degenerative effects and reduce intersymbol interference. It does it by approximating the inverse of channel's impulse response. The performance of the equalizer is critical to the successful operation of electronic systems.

The focus of this project is to investigate the feasibility of making such equalizer on an FPGA platform and then comparing its performance with analog DSP processor based system.

2 Theoretical Background

2.1 Transmission

Figure 1 presents a simplified block diagram of a digital communication system. The input data is applied to the *Modulator and transmitter*, which converts the serialized data sequence into a bandlimited analog waveform and translates it into frequency band suitable for transmission. As the signal propagates through the channel, it is affected by delay, attenuation and distortion in a frequency-dependent manner. On the other end, the receiver accepts the deteriorated signal and attempts to reconstruct the initial data sequence.

The digital data is usually presented in the form of a stream of binary data. Regardless if data is analog (audio and video) or digital (output of a computer) the goal is to transmit these data to the destination using the given communication channel. Depending

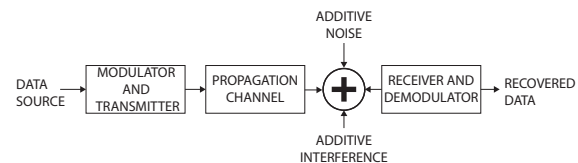


Figure 1: A simple model of a communications system (reproduced from [1]).

on a nature of communication channel transmission can suffer from various impairments, such as noise, attenuation, distortion, fading and interference.

To make the transmission the signal needs to be generated in a way that represents binary data, accommodates to channel characteristics such as bandwidth and impairments. Different channels have different characteristics impairments and as a result generated signal could be vastly different. The process of mapping a digital sequence to signals for transmission over a communication channel is called *digital modulation* or *digital signaling*. [2, p. 95]

2.1.1 Representation of digitally modulated signals

In order to send the digital sequence using the physical medium, the data must first be processed in a way that allows mapping the binary value to physical amplitude and phase. This process is called *data-mapping*.

The mapping between the binary sequence and the signal sequence to be transmitted over the channel is called *data-mapping*. Depending on modulation scheme, it could be either *memoryless* or *with memory*, resulting in corresponding modulation schemes.

In the memoryless scheme, such as PAM and QAM, the binary sequence is parsed into subsequences of length k , and each sequence is directly mapped into a corresponding set of waveforms $s_m(t)$, where $1 \leq m \leq 2^k$, regardless of previously transmitted signals. This is equivalent to a mapping from $M = 2^k$ messages to M possible signals. If we assume that these signals are at a signaling interval T_s , it means that in each second

$$R_s = \frac{1}{T_s} \quad (1)$$

symbols are transmitted, where R_s is called the *the signaling rate* or *symbol rate*. As each signal carries k bit of information, the *bit interval* T_b is given by

$$T_b = \frac{T_s}{k} = \frac{T}{\log_2 M} \quad (2)$$

and the *bit rate* R is given by

$$R = kR_s = R_s \log_2 M \quad (3)$$

If ε_m is the energy content of the waveform, then the average signal energy is

$$\varepsilon_{avg} = \sum_{m=1}^M p_m \varepsilon_m \quad (4)$$

where p_m is the probability of m th signal, or *message probability*. If message probabilities

are equal, or messages are *equiprobable*, $p_m = \frac{1}{M}$, and therefore

$$\epsilon_{avg} = \frac{1}{M} \sum_{m=1}^M \epsilon_m \quad (5)$$

If all signals have the same energy, then $\epsilon_m = \epsilon$ and $\epsilon_{avg} = \epsilon$. The average transmission energy per bit, when signals are equiprobable is

$$\epsilon_{bavg} = \frac{\epsilon_{avg}}{k} = \frac{\epsilon_{avg}}{\log_2 M} \quad (6)$$

To calculate the average energy power sent by the transmitter

$$P_{avg} = \frac{\epsilon_{bavg}}{T_b} = R \epsilon_{bavg} \quad (7)$$

Waveforms $s_m(t)$ that are used to transmit the signal over the communication channel can be in any form. However, as they are transmitted over bandwidth limited channel

2.1.2 Quadrature amplitude modulation

Quadrature amplitude modulation (QAM) is a memoryless analog, or digital modulation scheme.

The amplitude of two waves, 90° out-of-phase with each other (in quadrature) are changed (modulated or keyed) to represent the data signal. Amplitude modulating two carriers in quadrature can be equivalently

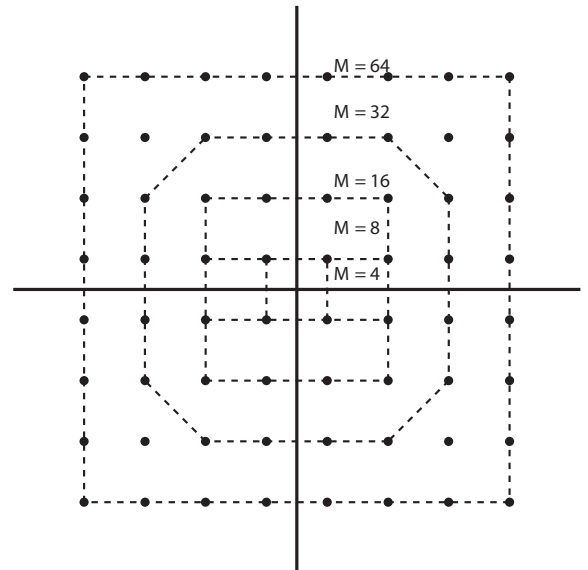


Figure 2: Several signal space diagrams(reproduced from [2]).

viewed as both amplitude modulating and phase modulating a single carrier. This is essentially achieved by mapping two separate k-bit symbols on two quadrature carriers $\cos 2\pi f_c t$ and $\sin 2\pi f_c t$ resulting in a signal waveform

$$\begin{aligned} s_m(t) &= \Re[(A_{mi} + jA_{mq})g(t)e^{j2\pi f_c t}] \\ &= A_{mi}g(t)\cos 2\pi f_c t - A_{mq}g(t)\sin 2\pi f_c t, \quad m = 1, 2, \dots, M \end{aligned} \quad (8)$$

where A_{mi} and A_{mq} quadrature carriers' signal amplitudes and $g(t)$ is the signal pulse.[2]

Another way to express the waveform is

$$s_m(t) = \Re[r_m e^{j\theta_m} e^{j2\pi f_c t}] = r_m \cos(2\pi f_c t + \theta_m) \quad (9)$$

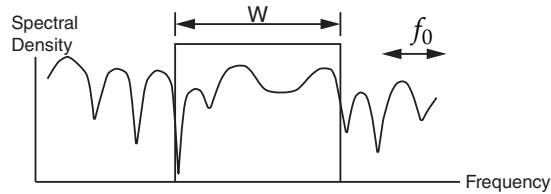
where $r_m = \sqrt{A_{mi}^2 + A_{mq}^2}$ and $\theta_m = \tan^{-1}(A_{mq}/A_{mi})$. It becomes apparent that QAM waveforms could be expressed as combined amplitude r_m and phase θ_m modulation.

In our case of transmitting discrete digital signals of length $(2m - 1 - M)$, $m = 1, 2, \dots, M$, the signal space diagram is rectangular, as shown in Figure 2.

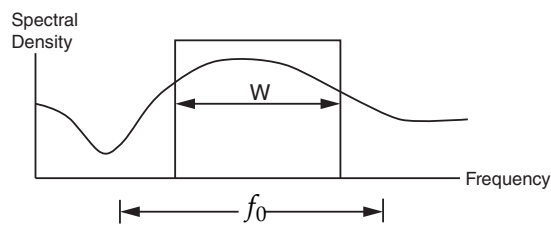
2.2 Channel

The general concept of fading channels was first modeled in 1950s and 1960s primarily targeting over-the-horizon communications covering a wide range of frequency bands. There are several models developed as of today such as additive-white-Gaussian-noise (AWGN) channel and Rayleigh fading channel. Ideal AWGN represents data sample, free of any intersymbol interference (ISI) being corrupted by statistically independent Gaussian noise samples. The main source of performance degradation is thermal noise generated in the receiver. This noise can be approximated as a white noise, having flat power density over the signal band. However, external interference received by the an-

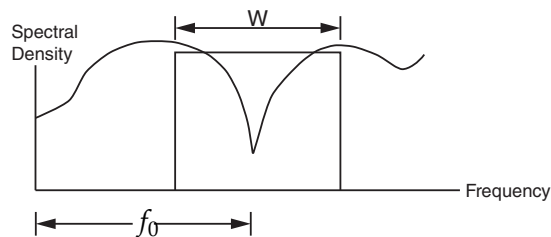
tenna can be more significant than the thermal noise. It can be characterized as having a broadband spectrum and quantized as antenna noise temperature. [3, 18-1]



(a) Typical Frequency-Selective Fading case ($f_0 < W$)



(b) Typical Flat-Fading case ($f_0 > W$)



(c) Null of Channel Frequency-Transfer Function occurs at
Signal Band Center ($f_0 > W$)

Figure 3: Relationships between the channel frequency-transfer function and a signal with bandwidth W (reproduced from [3]).

While AWGN is a good starting point in understanding basic principles, it is based on several assumptions that make it inaccurate in practical applications. In AWGN model it is assumed that signal attenuation vs. distance behaves as if propagation takes place over ideal free space, free of all the objects and obstacles, and the attenuation between transmitter and receiver behaves according to the inverse-square law.

However, for most practical cases the signal propagation takes place in the atmosphere, near the ground. The signal often travels from transmitter to receiver over multiple reflective paths, a phenomenon known as multipath propagation. This effect causes fluctuations in the received signal's amplitude, phase and angle of arrival.

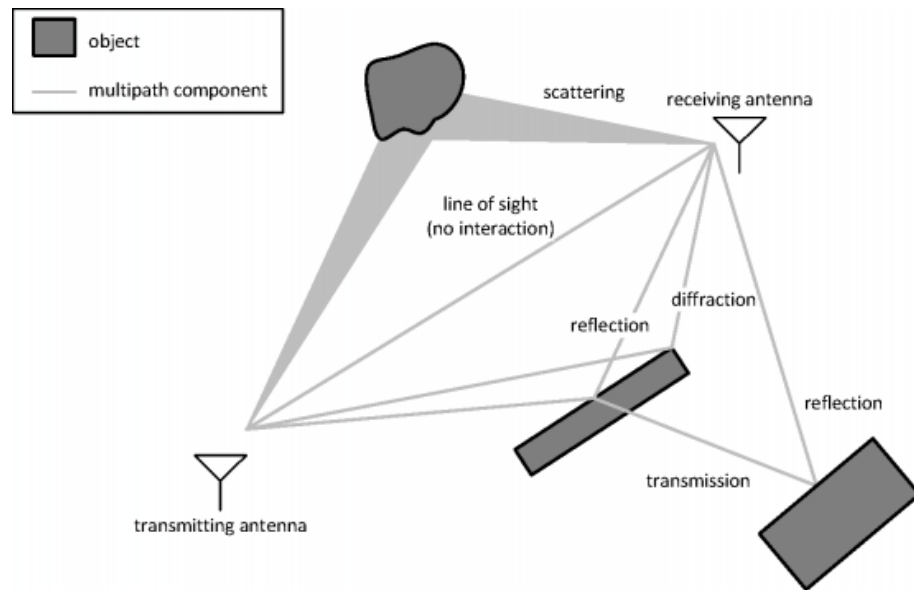


Figure 4: Multipath propagation (copied from [4]).

2.2.1 Rayleigh Fading Channel

There are 3 basic mechanisms that impact signal propagation: reflection, diffraction, and scattering.

Reflection occurs when a propagating electromagnetic wave impinges upon a smooth surface with very large dimensions compared to the RF signal's wavelength λ .

Diffraction occurs when the radio path between the transmitter and receiver is obstructed by a dense body with large dimensions compared to λ , causing secondary waves to be formed behind the obstructing body. Diffraction is a phenomenon that accounts for RF energy traveling from transmitter to receiver without line-of-sight between the two. It is often termed shadowing because the diffracted field can reach the receiver even when shadowed by an impenetrable obstruction.

Scattering occurs when a radio wave impinges on either a large rough surface of any surface whose dimensions are on the order of λ or less, causing the reflected energy to spread out (scatter) in all directions. In an urban environment, typical signal obstructions that yield scattering are lampposts, street signs, and foliage.

Figure 6 describes different channel fading manifestations.

To make a model of a realistic wireless channel, its impulse response is approximated and calculated. Then it could be implemented in a form of a tapped-delay filter.

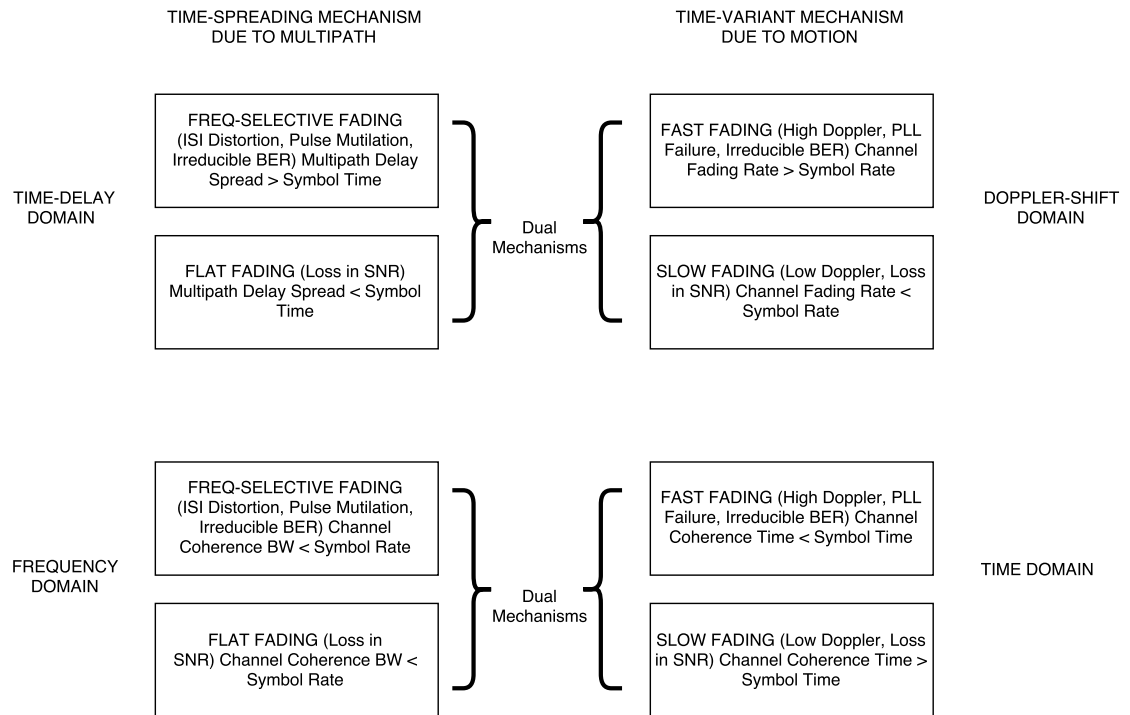


Figure 5: Small-scale fading: mechanisms, degradation categories, and effects (reproduced from [3]).

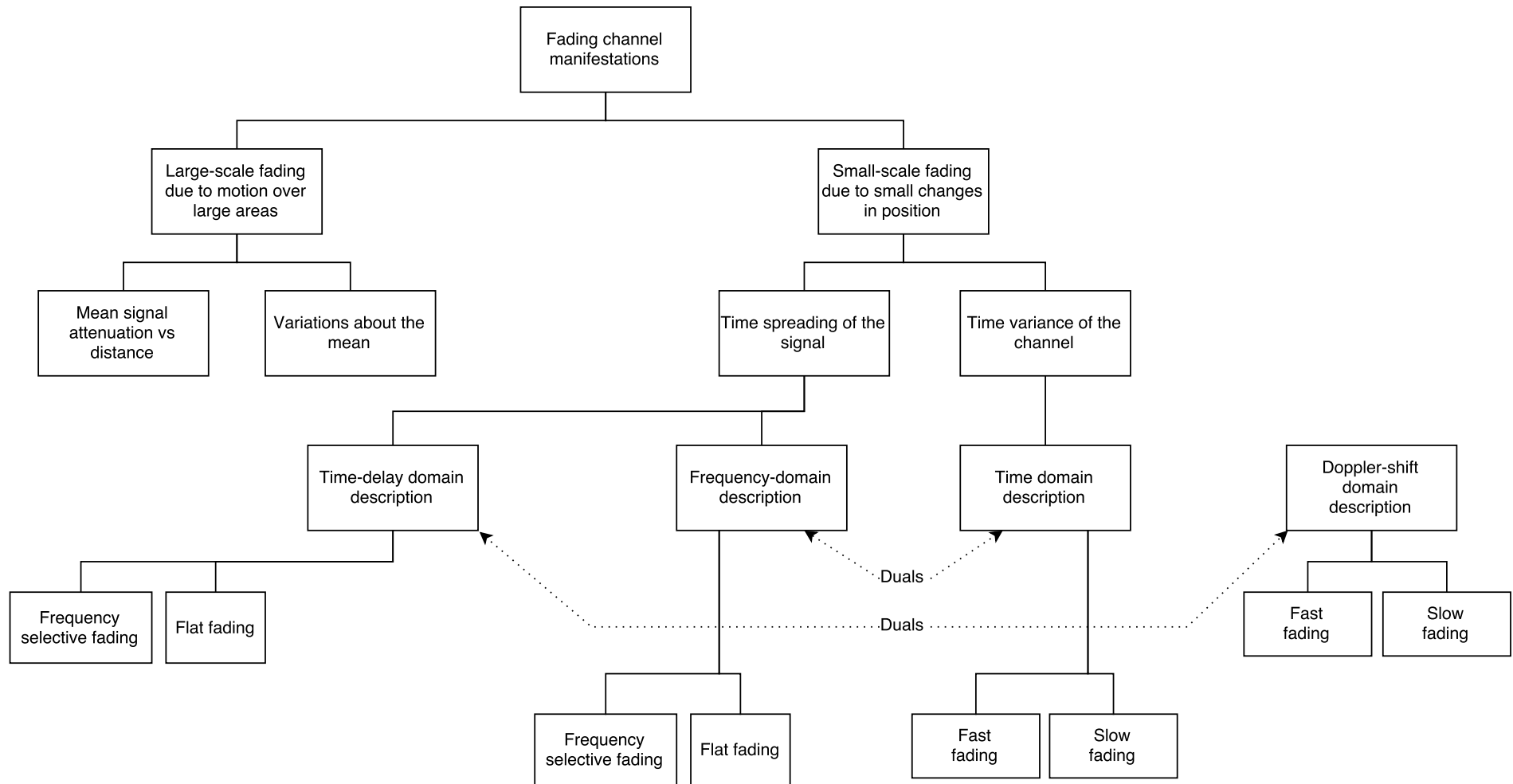


Figure 6: Fading channel manifestations (reproduced from Proakis [3]).

2.3 Reception

2.3.1 Demodulator

There are several design considerations when developing a demodulator. Essentially, the role of the demodulator is to: “(1) bandpass filter the incoming signal, (2) adjust the average input signal amplitude, (3) estimate and remove any carrier component, (4) equalize channel’s dispersive effects, (5) “time slice” the input signal to obtain pulse amplitude and phase measurements, (6) decide which pulse amplitude and phase pair was actually transmitted, and (7) convert that decision into associated bit pattern”. [1]

2.3.2 Equalizer

“The revolution in data communication technology can be dated from the invention of automatic and adaptive channel equalization in the late 1960s.” [7]

As the QAM was developed with the increase of bit-per-Hertz ratio for transmission, it also required the use of adaptive equalization to counteract the degenerative effects of the signal propagation. Coupled with the fact that equalizer tends to consume most of the receiver’s computational resources, great effort was put into finding an optimal tradeoff between cost and performance.

As the channel’s disruptive effects on the transmitted signal could be modeled as a linear filter, it is natural to use another linear filter to compensate or *equalize* these effects. If the channel characteristics are known and time-invariant, we only need to design it once. However, with our application, this is not the case. Not only the exact characteristics are unknown, they change with time. This means our filter must be equipped with control logic to learn the characteristics and, if necessary, to track and adjust to changes. This is the situation where adaptive equalizer is used.

There are two basic equalizer structures. The first one is called *linear equalizer* (Figure 7a) as the output is the linear combination of the received signal and its' delayed versions. The second one is called *decision feedback equalizer* (DFE) (Figure 7b), where the output is a combination of both filtered input signal and

filtered version of the nonlinear decision circuit's output. Deciding which type of equalizer to use depends on practical considerations of the task to solve. The biggest concern with DFE is that it is very complicated to pipeline the feedback loop in high-speed designs, hence linear equalizer was chosen.

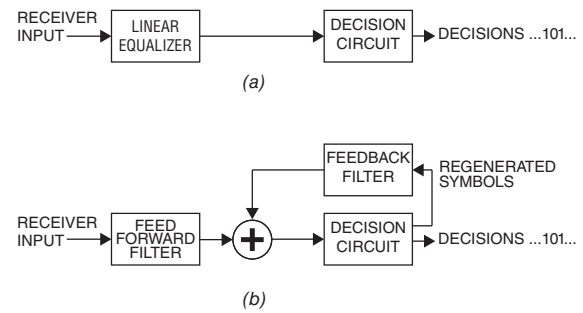


Figure 7: Linear and decision feedback equalizers (reproduced from [1]).

Early demodulator designs used T-space equalizers, meaning the input signal was sampled once every symbol. This required the timing of the sampled clock to be adjusted in a way, so that the samples are taken at the “top dead center” of received symbols.[8] It was an intuitive and computationally efficient way to design an equalizer. However, even though the pulses arrive at rate $1/T$, the actual bandwidth is 10 to 40 percent higher making it insufficient to satisfy the Nyquist theorem.[1] Thus, the use of a Fractionally Spaced Equalizer (FSE) was proposed.

The idea is to sample the received signals faster than the symbol rate f_B while keeping the output at f_B . While it is more computationally complex than T-spaced equalizer, it simplifies the overall design of demodulator and allows it to work at almost ideal level. The most commonly used input rate is simply $2 \cdot f_B$, making the filter tap spacing equal to $T/2$. [8]

The complexity of an equalizer is directly related to its length. Being encumbered by doubling the sample rate, it becomes even more important to keep the length as short as possible while still achieving desirable performance. Even though there is no good answer how to calculate such thing, 2 approaches became common. The first one is to simply build a prototype and test its performance against an actual channel. The second is to

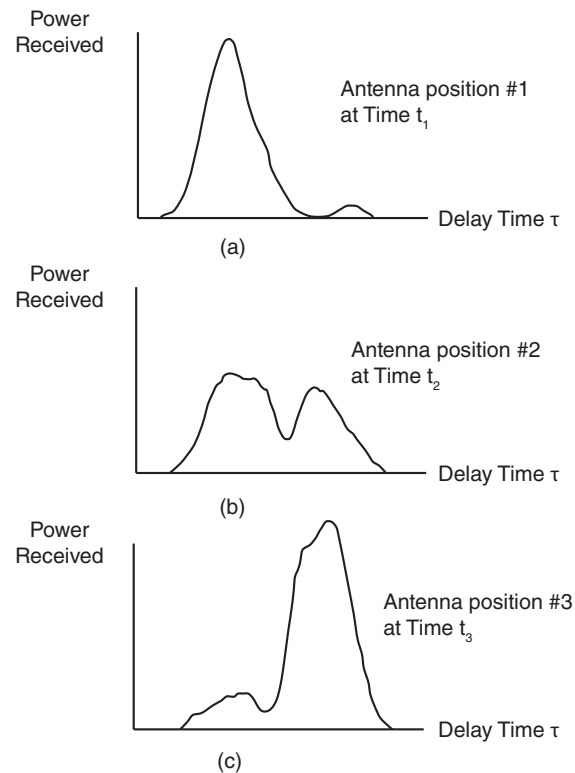


Figure 8: Response of a multipath channel to a narrow pulse vs. delay, as a function of antenna position (reproduced from [3]).

use some rule of thumb, based on idea that equalizer's convergent pulse response will approximate the inverse of the channel. [1]

2.3.3 Least Mean Squares filter

The LMS algorithm was first implemented in late 1950s by Widrow and Hoff in their study of a pattern-recognition machine. It is a simple and effective algorithm for the design of adaptive transversal filters. [9] The idea is to find filter coefficients that produces the least mean squares between the desired signal and the actual signal. LMS algorithm consist of 2 basic processes:

1. A *filtering process*, where the output of the transversal filter is computed and error estimation is generated
2. An *adaptive process*, where where tap weights of the filter are adjusted according to estimation error

This combination produces a feedback loop around the LMS algorithm. Its formal definition reads:

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \mu \mathbf{u}(n)[d(n) - \hat{\mathbf{w}}^H(n)\mathbf{u}(n)]^* \quad (10)$$

where $\hat{\mathbf{w}}$ is the tap-weight vector of the LMS filter, computed at n , \mathbf{u} is the tap-input vector, $d(n)$ is the desired response, and μ is the step-size parameter, the asterisk denotes complex conjugation and H denotes Hermitian transposition. An easier way to represent it would be with a block diagram 9.

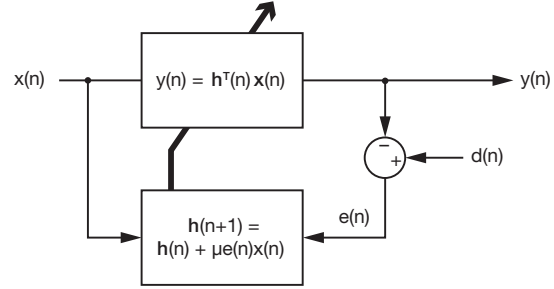


Figure 9: Adaptive LMS filter (copied from [5])

Here, the input signal $x(n)$ is given as a vector containing current sample followed by $N - 1$ samples. The output value of the FIR filter is a product of the input and the vector of N filter coefficients $h^T(n)$.

$$h(n) = \begin{bmatrix} h_1(n) \\ h_2(n) \\ \dots \\ h_N(n) \end{bmatrix}, \quad x(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \dots \\ x(n-N+1) \end{bmatrix} \quad (11)$$

$$\begin{aligned} y(n) &= h^T(n)x(n) \\ &= \sum_{i=1}^N h_i(n)x(n-i+1) \end{aligned} \quad (12)$$

$$e(n) = d(n) - y(n) \quad (13)$$

$$h(n+1) = h(n) + \mu e(n)x(n) \quad (14)$$

Error $e(n)$ is the difference between output $y(n)$ and the desired output $d(n)$. Based on that the filter coefficients are updated to minimize the output mean squared error $E[(d(n) - y(n))^2]$. Coefficient vector $h(n+1)$ is a sum of the current $h(n)$ with the weighted input vector $x(n)$, scaled with the error value $e(n)$ and adaptation rate μ . [5, 10]

3 Implementation

3.1 Workflow

While FPGA's can achieve incredible performance in streamlined tasks, conditional algorithms are achieved with a great cost. To alleviate that Xilinx designed a licensed IP softcore called Microblaze. It is a 32-bit RISC Harvard architecture soft processor core and is included in free version of Vivado. It is implemented entirely in the general-purpose memory and logic fabric of FPGA and therefore it allows for complete customization. In the system, softcore handles all the input/output transactions between the PC and the development board and prepares the data to be processed. After that, it starts the Adaptive Equalizer and feeds the data to the input. The interface between Ad. Eq. and the softcore is implemented with several GPIO buses and multiplexer logic. I and Q values are packed into a single variable using bitshift operations and then glue logic between AdEq and softcore splits it into two data buses. While this is a very simple and convenient solutions, it is also not optimal. It takes a great amount of processing time to do these operations, while the equalizer is idle. A better solution would be to implement AXI4 interface in the equalizer, which could be then accessed as a memory mapped interface.

3.2 Hardware

The choice of hardware provided a major challenge. There are hundreds of different configuration on the market starting from 100 euros up to several thousands. As the task on hand was calculation heavy, the FPGA chip has to have enough processing power. There are several ways to crunch number on FPGA. The first one is using inner fabric to generate adders and multipliers. There are several topologies that have different pros and cons. As this method is very area consuming, the manufacturers started to add

dedicated DSP blocks and multipliers to the chip to save the area.

Another essential component was the interface between FPGA board and PC. The industry standard is PCI Express providing 500 MB/s transfer rate and very easy protocol to implement.

After all the considerations, the choice was to use Digilent Nexys4 platform. It hosts Xilinx Artix-7 FPGA chip, 16 MByte of CellularRam and plethora of peripheral interfaces, including 10/100 Ethernet. The board was chosen for its impressive computational power, having 240 DSP blocks on the board. Each block has a pre-adder, 18x25 multiplier and 48 bit accumulator. Such architecture is great for any kind of transversal filters, allowing for parallelization and pipelining of computations.

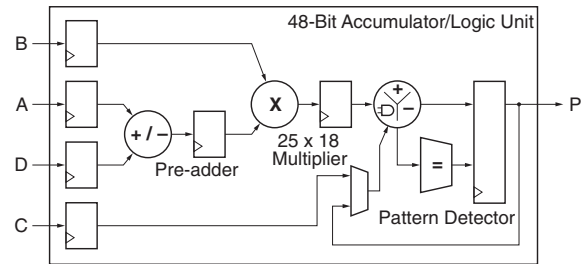


Figure 10: Basic DSP48E1 Slice Functionality (copied from [6]).

3.3 Software

To provide an optimal solution in timely manner, a whole set of software was used. The top level level design was created in Xilinx Vivado Design Suite. It allows to use Xilinx IPs for the general tasks as well as packaging your own IPs from VHDL or Verilog code. Filter design was done in System Generator. It is an add-on to Simulink made by Xilinx that allows to combine design, synthesis, and verification in a single tool. After the filter is synthesized, it is packaged in IP package using Xilinx Vivado and integrated into the system. This all allows for a modular based design, where you start with a simple proof-of-concept and progress to a more complicated design, debugging and verifying the design with each step.

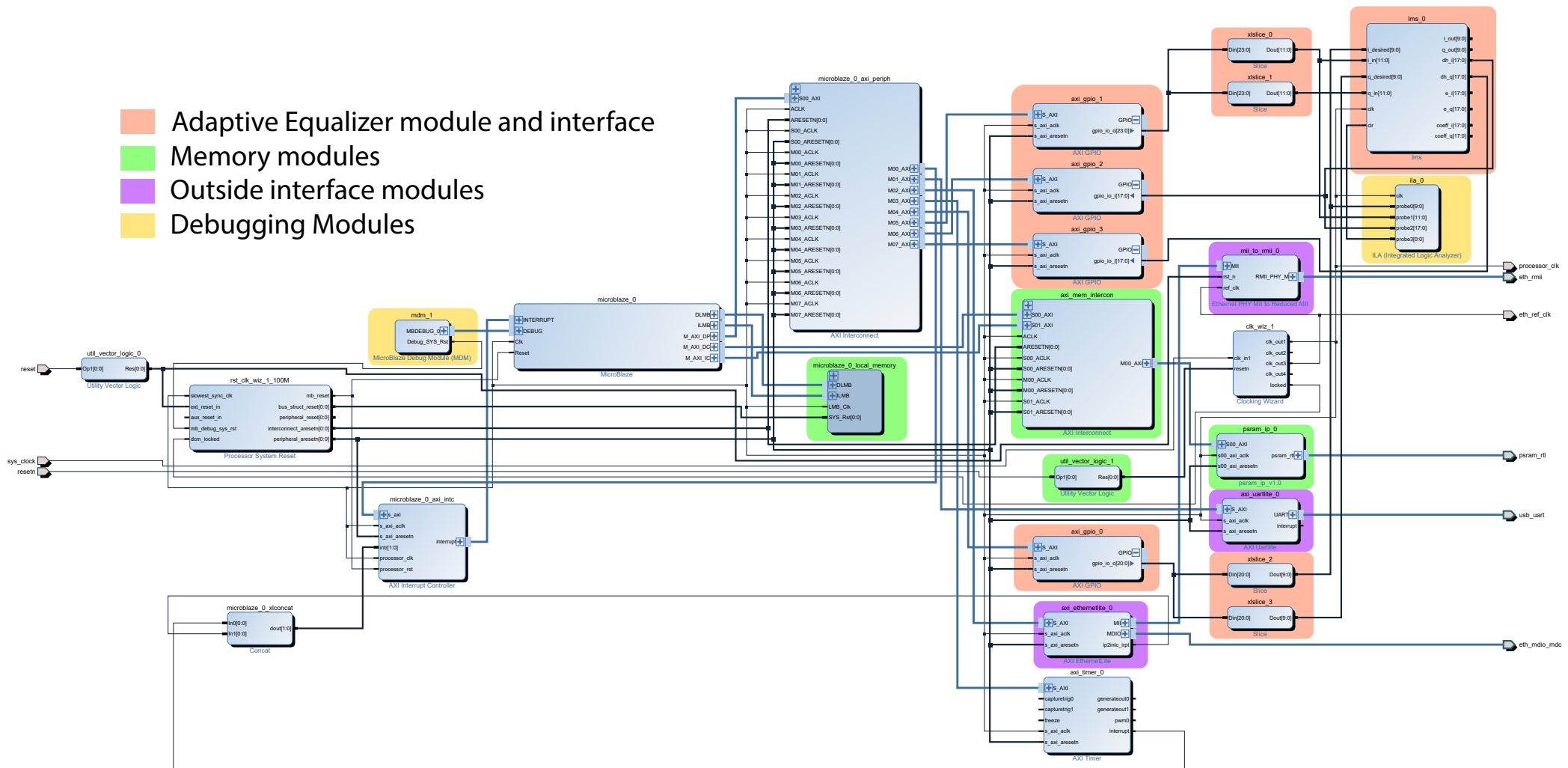


Figure 11: Complete system

3.4 Implementation

Fractionally spaced equalizer receives K samples to produce 1 output samples and update it weights. In this implementation this number is 2 meaning the output sample rate is $1/T$ where input sample rate is K/T . The weight-updating occurs at the output rate, which is the slower rate.

Initial design and simulation of the equalizer were done in System Generator. The software works in cooperation with Simulink and allows to automatically make testbenches for VHDL simulation. The design is heavily based on a standard implementation provided by Xilinx Inc. Figure 12 presents top view on the equalizer. *QAM16 source* generates I and Q parts of the signal, which are merged into a complex number. In these numbers are convoluted with a sinc filter and interpolated to produce band limitation. In *FIR Channel Model* FIR filter with coefficients $[1 + 0.1j \quad 0.2 \quad 0.2j]$ is applied to the signal, simulating a channel. The unmodified signal from *QAM16 source* is fed into the equalizer as a training sequence.

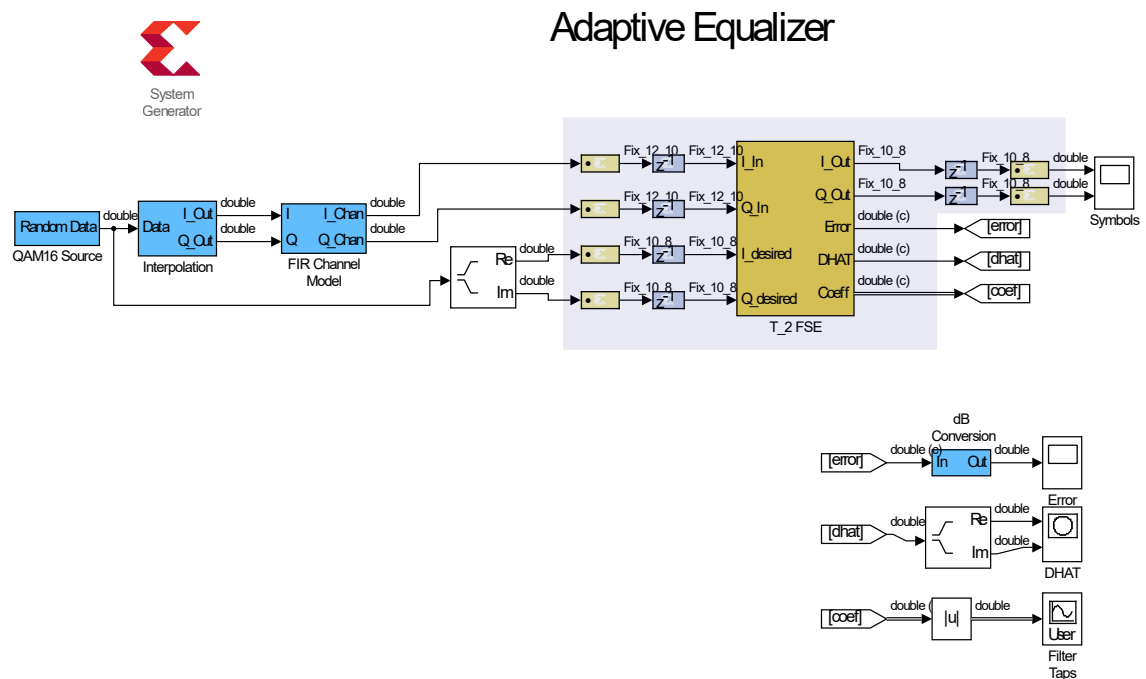


Figure 12: Top view of System Generator

Figure 13 represents the equalizer. The input signal is fed into the filter. The output of the

filter is later compared with the desired value and the error (Listing 1) is calculated.

```

1  function [mu_e_i, e_i, e_q, mu_e_q] = lmsErr(sym_i, filt_i, sym_q, filt_q, mu)
2
3  persistent r0, r0 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
4  persistent r1, r1 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
5  persistent r2, r2 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
6  persistent r3, r3 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
7  persistent r4, r4 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
8  persistent r5, r5 = xl_state(mu, {xISigned, 18, 15, xIRound, xIWrap});
9
10 mu = xfix({xISigned, 18, 15, xIRound, xISaturate}, mu);
11
12 e_i = r4;
13 r4 = sym_i - filt_i;
14
15 e_q = r5;
16 r5 = sym_q - filt_q;
17
18 mu_e_i = r0;
19 r0 = r1;
20 r1 = r4 * mu;
21
22 mu_e_q = r2;
23 r2 = r3;
24 r3 = -r5 * mu;

```

Listing 1: Error Calculation

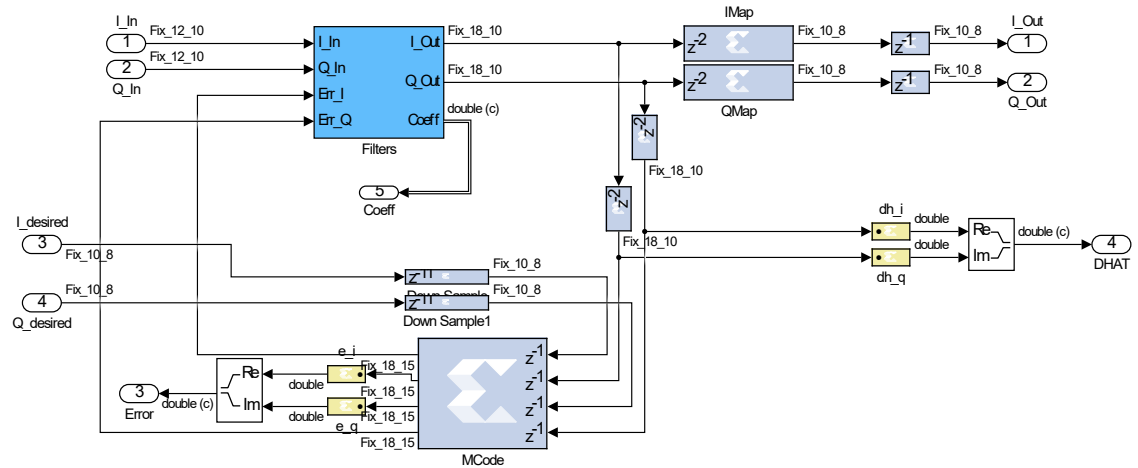


Figure 13: Fractionally spaced equalizer

The implementation of the filter algorithm is represented in Figure 14. Both LMS structures are identical, the input data is downsampled at the ratio of 2. On the top LMS the input data is also delayed by 1 sample so that the filters can work in parallel.

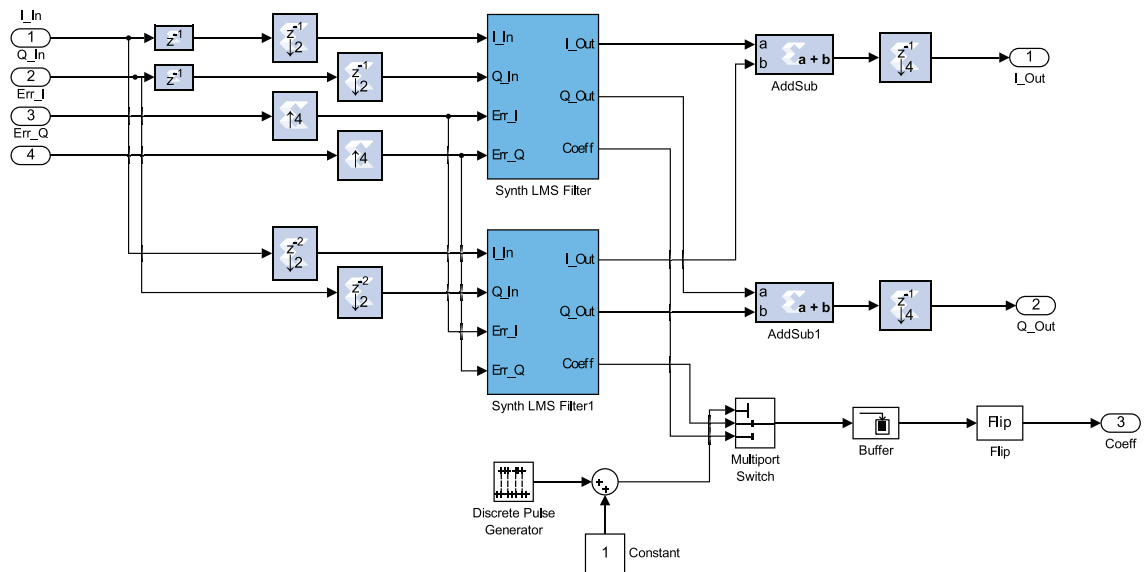


Figure 14: Adaptive Filter top view

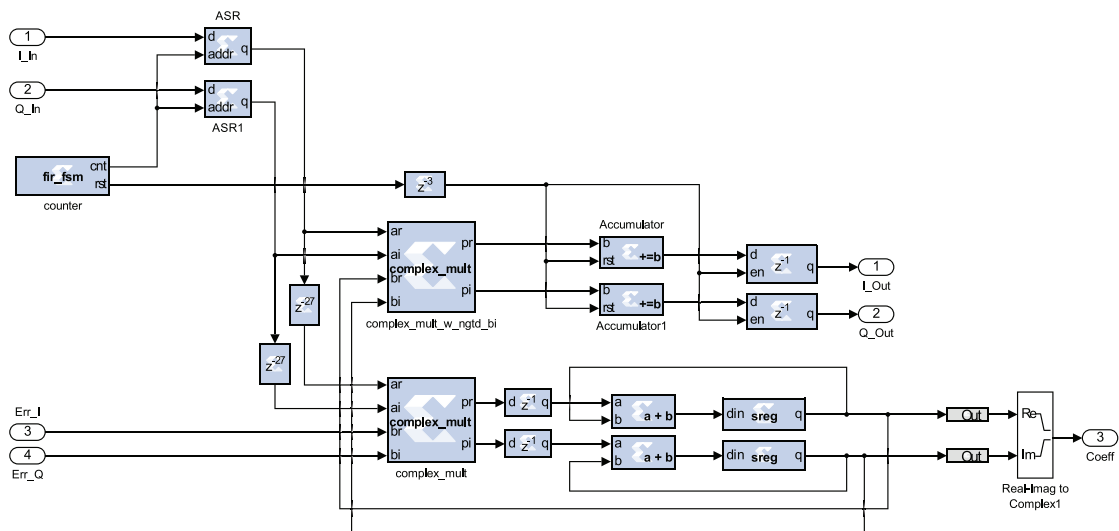


Figure 15: LMS Filter

The inner workings of LMS filter are a bit trickier. *ASR* on the left side is shift register of length 4, *fir_fsm* is a 2 bit counter, counting from 0 to 4. What it does it synchronizes the whole system so that each value in the buffer is multiplied with corresponding filter coefficient. As there are 4 values in the buffer and there are 2 structures like that in parallel, it means we have an 8-tap filter. The sum of all the values is kept in the *Accumulator*. When the counter goes through the whole cycle, the *rst* bit is set, which stores the data

in the output register and nullifies the accumulator.

4 Results

To produce repeatable results a test bench was designed. The idea is to eliminate as many unnecessary elements as possible while keeping the system running smoothly thus decreasing the invariance. For a precise measurement, a timer module was added on FPGA fabric and connected to the processor via AXI-LITE bus. The first test is run on 10 000 samples to evaluate convergence rate and overall error reduction.

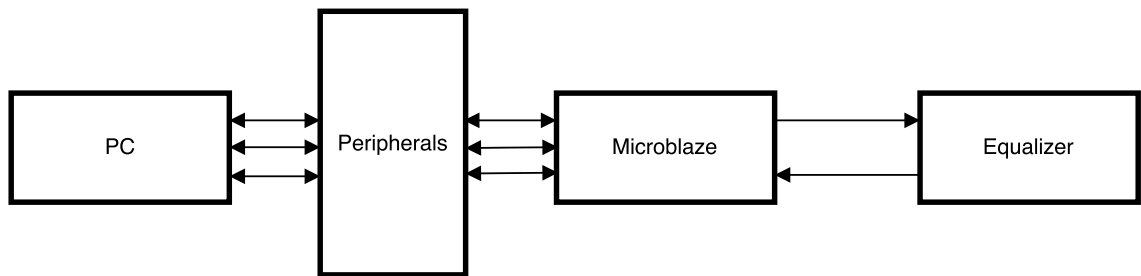


Figure 16: Simplified diagram of the test environment

As demonstrated in Figure 17, it takes approximately 550 iterations for the filter to converge. After that it stays at the error magnitude of 0.1. For the benchmark, an n -long sequence of 12 bit values is prepared for both I and Q channels, where n is 50, 100 or 200. Then, the timer is initialized and started. Benchmarks for every n is repeated 100 times. Every time, the value of the timer is read before and after the benchmark and then subtracted. As the microprocessor runs at 100 MHz, the value from the timer is then divided by $100 \cdot 10^6$. The code could be found in the Appendix 1.

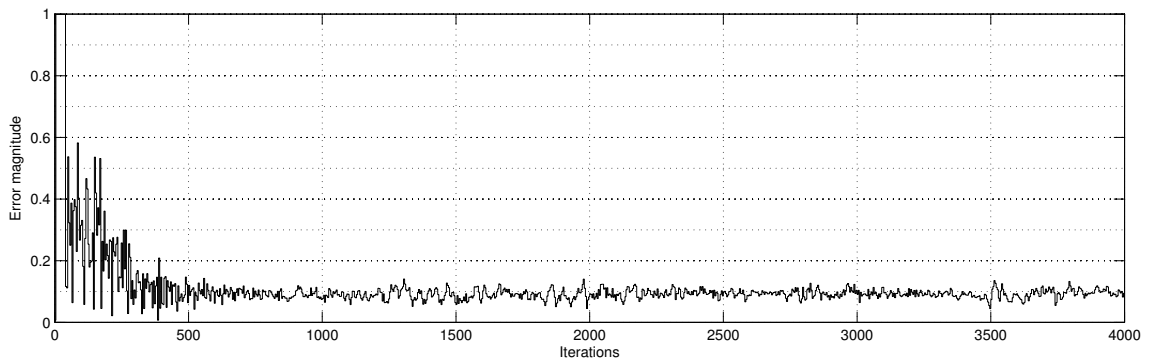


Figure 17: Equalizer error at $\mu=0.05$

Table 1 shows the execution time in respect to the amount of samples in one frame.

As you can see from the results, first two executions add fixed overhead of 434 and 33 clocks. On the third and all the consecutive runs the results are surprisingly consistent. It also becomes apparent there is a slight inverse correlation between the size of the frame and the average time required to process each sample in the frame.

#	n = 50	n = 100	n = 200
1	1444	2744	5344
2	1043	2343	4943
3	1010	2310	4910
4	1010	2310	4910
5	1010	2310	4910
...
95	1010	2310	4910
96	1010	2310	4910
97	1010	2310	4910
98	1010	2310	4910
99	1010	2310	4910
100	1010	2310	4910
average clock per value	20,2934	23,1467	24,57335
average time per value (ns)	202,934	231,467	245,7335

Table 1: Execution time (lower is better) at different sample sizes

To compare results with corresponding work from [5, p. 57, Table 6], the benchmark was modified so that the sample size is the same 64 samples. The results were the integrated into Table 2. The comparison clearly shows the massive advantage FPGA has over the DSP implementation. There are several reasons to this.

	Total average, ns	Max, ns	Min, ns	Records
FPGA FSE	13 786,7	18 080	13 740	100
LMS	552 296,42	513 860	552 544	2856
NLMS	679 089,46	641 005	679 325	1070
Complex LMS	1 174 887,12	1 098 390	1 175 245	2208
Complex NLMS	1 298 153,61	1 221 652	1 298 738	818
Complex LMS 2 cores	2 289 930,02	2 290 858	2 289 170	3532
Complex NLMS 2+1 cores	4 731 336,67	4 818 911	4 616 363	2477

Table 2: Algorithm execution benchmarks

The biggest impact is caused in difference in architecture. The flexibility of FPGA allows for concurrent and pipelined computation of the algorithm. The data flow could be divided into smaller streams and processed in parallel. Having a configurable hardware meaning the whole system could be streamlined. The LMS module is connected directly to the microprocessor and requires very little overhead to use. There is no loss because of context switching and multicore synchronization. The DSP implementation also uses

floating point calculations. While in theory it could provide better accuracy, it is also computationally more expensive and slower.

Bibliography

- 1 Treichler JR, Fijalkow I, Johnson CR. Fractionally Spaced Equalizers. IEEE Signal Processing Magazine. 1996;13. 4, 2, 10, 11, 12
- 2 Proakis J, Salehi M. Digital Communications. 5th ed. McGraw Hill; 2008. 4, 2, 5
- 3 Gibson J. The mobile communications handbook. 2nd ed. CRC Press LLC; 1999. 4, 6, 8, 9, 12
- 4 Ghent University. Physical radio channel models;. Accessed: 15.01.2016. <http://www.wica.intec.ugent.be/research/propagation/physical-radio-channel-models>. 4, 7
- 5 Kempf I. Adaptive Channel Equalization: Multicore Processor Implementation. Metropolia University of Applied Sciences; 2015. 4, 13, 14, 22
- 6 Xilinx, Inc. 7 Series DSP48E1 User Guide; 2014. 4, 16
- 7 Gitlin RD, Hayes JF, Weinstein SB. Data Communication Principles. Plenum Press; 1992. 10
- 8 Bingham JAC. The Theory and Practice of Modem Design. Wiley Interscience; 1988. 11
- 9 Haykin S, Widrow B. Least-Mean-Square Adaptive Filters. Wiley Interscience; 2003. 12

10 Haykin S. Adaptive Filter Theory. 3rd ed. Prentice Hall; 1995. 14

1 Benchmark Code

```
1
2  #include <stdio.h>
3  #include "platform.h"
4  #include <limits.h>
5  #include "fsl.h"
6  #include "xtmrctr.h"
7  #include "xparameters.h"
8
9  #include "test_values.h"
10 #define COUNT 64
11
12 void print(char *str);
13
14 int main()
15 {
16     init_platform();
17
18     print("START\n\r");
19     int out_i[COUNT];
20     int out_q[COUNT];
21     //Initializer timer
22     XTmrCtr TmrCtrInstancePtr;
23     XStatus Status = XTmrCtr_Initialize(&TmrCtrInstancePtr, XPAR_TMRCTR_0_DEVICE_ID);
24     if (Status != XST_SUCCESS) {
25         return XST_FAILURE;
26     }
27
28     //Timer test
29     Status = XTmrCtr_SelfTest(&TmrCtrInstancePtr, 0);
30     if (Status != XST_SUCCESS) {
31         return XST_FAILURE;
32     }
33     //Load timer config
34     XTmrCtr_SetOptions(&TmrCtrInstancePtr, 0, XTC_AUTO_RELOAD_OPTION);
35     volatile u32 TValue1;
36     volatile u32 TValue2;
37
38     TValue1 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);
39     XTmrCtr_Start(&TmrCtrInstancePtr, 0);
40
41     int TimerValues[100];
42
43     int c, i;
44     for(i = 0; i < 100; i++){
45         TValue1 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);
46         for(c = 0; c < COUNT; c++){
47             //load values into equalizer
48             putfsIx(in_i[c], 0, FSL_DEFAULT);
49             putfsIx(in_q[c], 1, FSL_DEFAULT);
50             // get values from equalizer
```

```
51         getfslx(out_i[c], 0, FSL_DEFAULT);
52         getfslx(out_q[c], 1, FSL_DEFAULT);
53     }
54     TValue2 = XTmrCtr_GetValue(&TmrCtrInstancePtr, 0);
55     TimerValues[i] = TValue2-TValue1;
56     //send timer values
57     xil_printf("%d\r\n", TimerValues[i]);
58 }
59
60 //send test results
61 xil_printf("out_i:\r\n");
62 for (c = 0; c < COUNT; c++)
63     xil_printf("%d\r\n", out_i[c]);
64
65 xil_printf("out_q:\r\n");
66 for (c = 0; c < COUNT; c++)
67     xil_printf("%d\r\n", out_q[c]);
68
69 cleanup_platform();
70 return 0;
71 }
```

Listing 2: Benchmark Code